HD

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/748,785	12/22/2000	Andrew Cofler	S1022/8583	3330

7590 01/13/2006

James H. Morris
c/o Wolf, Greenfield & Sacks, P.C.
Federal Reserve Plaza
600 Atlantic Avenue
Boston, MA 02210-2211

EXAMINER	
HUISMAN, DAVID J	

ART UNIT	PAPER NUMBER
2183	

DATE MAILED: 01/13/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	09/748,785	COFLER ET AL.	
	Examiner	Art Unit	
	David J. Huisman	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *19 October 2005*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-18* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-18* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *30 June 2004* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☒ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☒ All b)☐ Some * c)☐ None of:

1.☒ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-18 have been examined.

Papers Submitted

2. It is hereby acknowledged that the following papers have been received and placed of record in the file: Amendment and Extension of Time as received on 10/19/2005.

Maintained Rejections

3. Applicant has failed to overcome the prior art rejections set forth in the previous Office Action for claims 1-15. Consequently, these rejections are respectfully maintained by the examiner and are copied below for applicant's convenience. A new grounds of rejection has been applied to claims 16-18, using all previously applied art, since claim 16 has been amended and the scope of the claim has changed.

Claim Rejections - 35 USC § 112

4. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

5. Claims 16-18 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

6. Claim 16 recites the limitation "the execution unit" in lines 14-15. There is insufficient antecedent basis for this limitation in the claim. Please replace "the execution unit" with --the at least one execution unit."

Claim Rejections - 35 USC § 103

7. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

8. Claims 1-5, 8-10, 12, 14, and 16 are rejected under 35 U.S.C. 103(a) as being unpatentable over Song, U.S. Patent No. 5,546,599 (as applied in the previous Office Action), in view of Grochowski et al., U.S. Patent No. 6,353,883 (as applied in the previous Office Action and herein referred to as Grochowski).

9. Referring to claim 1, Song has taught a computer system for executing instructions comprising:

a) a fetch unit for fetching instructions to be executed. See Fig.3, and note that instructions are fetched from instruction cache 14 into instruction buffer 70.

b) a decode unit for decoding said instructions. See Fig.3, component 72.

c) at least one pipelined execution unit for executing decoded instructions. From Fig.3, it should be realized that decoded instructions are eventually dispatched to execution units (which are shown in Fig.1). In addition, see column 4, lines 11-16, and note that that the execution units

exist within a multi-stage pipeline and that the execution units may require multiple cycles themselves (specifically, in Fig.13 and 15).

d) an emulation unit including control circuitry which cooperates with the decode unit to selectively control the decode unit to implement a precise watch or a non-precise watch on detection of a breakpoint caused by a fetched instruction having a specified program count or a fetched instruction having a specified opcode, wherein according to a non-precise watch, the instruction causing the breakpoint and subsequent instructions are not allowed to enter the execution unit. See column 9, line 48, to column 10, line 7. Note that a breakpoint/exception is caused by an illegal instruction, for instance. An illegal instruction has an opcode that is not in the set of legal opcodes. Consequently, the breakpoint is caused by a specified (illegal) opcode. In this case (precise), it and its subsequent instructions are dispatched to reservation stations of an execution unit. See Fig.2, components 50a and 50b. However, the instructions are not allowed to enter the true execution unit (execution logic) 54 for execution if a breakpoint was caused (this is determined by checking the EOK bit). It should be realized that in the claim, applicant has not defined what it means to not enter an execution unit. For purposes of this examination, the execution unit is the actual logic 54 which performs execution. It should also be noted from column 10, lines 26-28, that in one technique, the breakpoint instruction is never dispatched to the execution unit. Both of these aforementioned teachings read on applicant's claims. Furthermore, in column 10, lines 12-16, it is disclosed that the breakpoint/exception is processed after all instructions preceding the breakpoint are complete. As a result, it should be realized that the instructions subsequent to the breakpoint instruction are not executed (don't enter the execution unit) until the exception is processed. On the other hand, note from column

22, lines 16-29, that the system also operates in a non-precise (imprecise) exception mode. More specifically, the breakpoint instruction causing the exception (caused at least in part by an instruction having a floating-point opcode) along with subsequent instructions are allowed to proceed through the pipeline so that increased performance is achieved.

e) Song has not taught that:

- the executed instructions are predicated instructions, wherein each instruction includes a guard, the value of which determines whether or not that instruction is executed. However, Grochowski has taught the concept of executing predicated instruction having guards. See Fig.6, step 610, and Fig.1 (note that p2 is a predicate/guard).
- the execution unit is associated with a guard register file holding values of the guards to allow resolution of the guards to be made. However, Grochowski has taught such a concept. See Fig.3A, component 300, and column 2, lines 65-67. Note that the predicate table (register file), is updated with actual predicate values, which are in turn used as predictions for speculatively executed instructions.
- instructions are supplied to the execution unit while guard resolution in said execution pipeline is awaited. However, Grochowski has taught such a concept. See column 3, lines 4-35. More specifically, predicated instructions are allowed to progress through the pipeline before its corresponding guard is resolved.

A person of ordinary skill in the art would have recognized that by implementing predicated instructions with predicate prediction within Song, a) conditional branches could be eliminated, thereby reducing the amount of instructions required in the instruction set, and b)

predicated instructions would be speculatively executed (i.e., executed before the corresponding guard is resolved), thereby maximizing throughput by executing predicated instructions as soon as possible. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Song to include predicated instructions and predicate prediction, as taught by Grochowski. It should be noted that applicant has not tied together the guard concept and the watch-mode concepts. Based on the claim language, applicant just has a mode where guard resolution is awaited, and this would happen in any guarded/predicated system.

10. Referring to claim 2, Song in view of Grochowski has taught a computer system as described in claim 1. Song has further taught that the system is implemented on a single chip. See Fig.1, component 10, and column 2, lines 47-48.

11. Referring to claim 3, Song in view of Grochowski has taught a computer system as described in claim 1. Song has further taught a program memory for holding said instructions to be executed. See Fig.1 and column 2, lines 60-61.

12. Referring to claim 4, Song in view of Grochowski has taught a computer system as described in claim 1. Song has not explicitly taught that the emulation unit is associated with an emulation program memory which holds debug code which is executed in a debug mode. However, recall that Song has taught handling exceptions (see Fig.3 and column 8, lines 10-13). As is known in the art, when an exception/interrupt is triggered during execution of a program, a handling routine must be invoked in order to correct the error associated with the exception/interrupt before execution of the main program may resume. According to "The American Heritage® Dictionary of the English Language, 3rd Edition, 1992," the word "debug" is defined as "to search for and eliminate malfunctioning elements or errors in." Consequently,

when an error occurs in an executing program in Song, an exception will be triggered. A handling routine (debug code) is invoked in order to search for the problem causing the exception and eliminate the error. This process would be considered debug mode. In addition, it should be realized that the debug code must be stored somewhere (emulation program memory). As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to implement an emulation program memory which holds debug code which is executed in a debug mode in Song since this feature is well known, accepted, and expected in the art of exceptions and exception handling.

13. Referring to claim 5, Song in view of Grochowski has taught a computer system as described in claim 1. Song in view of Grochowski has further taught that when the emulation unit is in a precise watch mode, it is operable to issue a request to the execution pipeline for guard resolution, the guard resolution being transmitted to the control circuitry of the emulation unit which is responsive thereto to control operation of the decode unit. Note from column 3, lines 25-35, of Grochowski, that a predicated instruction is prevented from executing until the guard is resolved. This is advantageous in that a time-expensive recovery is not required if the guard is improperly predicted. Therefore, when Song is in precise watch mode, Grochowski has taught that it is beneficial to issue a request to the execution pipeline for guard resolution for the aforementioned reasons. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to issue a request to the execution pipeline for guard resolution during precise watch mode in Song.

14. Referring to claim 8, Song in view of Grochowski has taught a computer system as described in claim 1. Song has further taught a microinstruction generator which receives

instructions from the decode unit and supplies microinstructions to the execution pipeline. See Fig.3, component 74, and column 7, lines 23-26, and note that up to 4 microinstructions that are to be dispatched are determined (aka produced or generated). In addition, Grochowski has taught that said microinstructions include fields for holding respective guards to be resolved. For instance, see column 5, lines 36-41, and note that the predicate is extracted from the instruction. Therefore, there must be a field within the instruction which specifies the predicate.

15. Referring to claim 9, Song in view of Grochowski has taught a computer system as described in claim 1. Song has further taught a plurality of parallel pipelined execution units, including at least two data unit pipelines for executing data processing instructions and at least two address unit pipelines for executing memory access instructions. From Fig.1, it should be seen that floating-point unit (30) and fixed point unit A (22) are data units in that they will execute floating and fixed point instructions (such as add, subtract, mult, etc.), and load/store unit (28) and complex fixed-point unit are address units in that the load/store unit will execute loads and stores, which access memory, and complex fixed point unit also accesses memory (general and special purpose registers, which are forms of memory). Finally, it should be realized that to make separable is not generally a patentable feature or would be an obvious improvement. More specifically, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Song's load/store unit into a separate load unit and store unit, thereby yielding two address units.

16. Referring to claim 10, Song has taught a method of debugging an on-chip processor which is arranged to execute instructions, the method comprising:

a) fetching instructions to be executed. See Fig.3, and note that instructions are fetched from instruction cache 14 into instruction buffer 70.

b) decoding said instructions. See Fig.3, component 72.

c) executing decoded instructions, said executing step including resolving values of the guards of the instructions. From Fig.3, it should be realized that decoded instructions are eventually dispatched to execution units (which are shown in Fig.1).

d) detecting instructions which have a debug effect based on a program count or an opcode of the instructions and acting on said instructions in dependence on whether the processor is in a precise watch mode or a non-precise watch mode wherein, according to a precise watch mode, an instruction having a debug effect and subsequent instructions are not executed and according to a non-precise watch mode, the instruction and subsequent instructions are supplied and executed normally. See column 9, line 48, to column 10, line 7. Note that a breakpoint/exception is caused by an illegal instruction, for instance. An illegal instruction has an opcode that is not in the set of legal opcodes. Consequently, the breakpoint is based on an opcode of the instruction. In this case (precise), it and its subsequent instructions are dispatched to reservation stations of an execution unit. See Fig.2, components 50a and 50b. However, the instructions are not allowed to enter the true execution unit (execution logic) 54 for execution if a breakpoint was caused (this is determined by checking the EOK bit). It should be realized that in the claim, applicant has not defined what it means to not enter an execution unit. For purposes of this examination, the execution unit is the actual logic 54 which performs execution. It should also be noted from column 10, lines 26-28, that in one technique, the breakpoint instruction is never dispatched to the execution unit. Both of these aforementioned teachings read on

applicant's claims. Furthermore, in column 10, lines 12-16, it is disclosed that the breakpoint/exception is processed after all instructions preceding the breakpoint are complete. As a result, it should be realized that the instructions subsequent to the breakpoint instruction are not executed (don't enter the execution unit) until the exception is processed. On the other hand, note from column 22, lines 16-29, that the system also operates in a non-precise (imprecise) exception mode. More specifically, the breakpoint instruction causing the exception (caused at least in part by an instruction having a floating-point opcode) along with subsequent instructions are allowed to proceed through the pipeline so that increased performance is achieved.

e) Song has not taught that:

- the executed instructions are predicated instructions, wherein each instruction includes a guard, the value of which determines whether or not that instruction is executed. However, Grochowski has taught the concept of executing predicated instruction having guards. See Fig.6, step 610, and Fig.1 (note that p2 is a predicate/guard).
- Said executing step includes resolving values of the guards of the instructions. However, Grochowski has taught such a concept. See Fig.1. Note that by executing the COMPARE instruction, the guard p2 will be resolved.
- instructions are supplied to the execution unit while guard resolution in said execution pipeline is awaited. However, Grochowski has taught such a concept. See column 3, lines 4-35. More specifically, predicated instructions are allowed to progress through the pipeline before its corresponding guard is resolved.

A person of ordinary skill in the art would have recognized that by implementing predicated instructions with predicate prediction within Song, a) conditional branches could be eliminated, thereby reducing the amount of instructions required in the instruction set, and b) predicated instructions would be speculatively executed (i.e., executed before the corresponding guard is resolved), thereby maximizing throughput by executing predicated instructions as soon as possible. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Song to include predicated instructions and predicate prediction, as taught by Grochowski. It should be noted that applicant has not tied together the guard concept and the watch-mode concepts. Based on the claim language, applicant just has a mode where guard resolution is awaited, and this would happen in any guarded/predicated system.

It should be further realized that according to "The American Heritage® Dictionary of the English Language, 3rd Edition, 1992," the word "debug" is defined as "to search for and eliminate malfunctioning elements or errors in." Consequently, when an error occurs in an executing program in Song, an exception will be triggered. A handling routine (debug code) is invoked in order to search for the problem causing the exception and eliminate the error. This process would be considered debugging.

17. Referring to claim 12, Song in view of Grochowski has taught a method as described in claim 10. Song in view of Grochowski has further taught that breakpoints are detected at instructions having certain opcodes. From column 9, line 48, to column 10, line 7, it should be realized that illegal instructions (with illegal opcodes) cause breakpoints/exceptions. Also, from column 22, lines 16-29, breakpoints result from floating-point errors. So the breakpoint is partly caused by the instruction being a floating-point instruction (having a floating-point opcode).

18. Referring to claim 14, Song in view of Grochowski has taught a method as described in claim 12. Furthermore, recall that Song has taught handling exceptions (see Fig.3 and column 8, lines 10-13). As is known in the art, when an exception/interrupt is triggered during execution of a program, a handling routine must be invoked in order to correct the error associated with the exception/interrupt before execution of the main program may resume. According to "The American Heritage® Dictionary of the English Language, 3rd Edition, 1992," the word "debug" is defined as "to search for and eliminate malfunctioning elements or errors in." Consequently, when an error occurs in an executing program in Song, an exception will be triggered. A handling routine (debug code) is invoked and executed by the processor in order to search for the problem causing the exception and eliminate the error. This process would be considered a debug mode.

19. Referring to claim 16, Song has taught a computer system for executing instructions comprising:

a) a fetch unit for fetching instructions to be executed. See Fig.3, and note that instructions are fetched from instruction cache 14 into instruction buffer 70.

b) a decode unit for decoding said instructions. See Fig.3, component 72.

c) at least one pipelined execution unit for executing decoded instructions. From Fig.3, it should be realized that decoded instructions are eventually dispatched to execution units (which are shown in Fig.1). In addition, see column 4, lines 11-16, and note that that the execution units exist within a multi-stage pipeline and that the execution units may require multiple cycles themselves (specifically, in Fig.13 and 15).

d) an emulation unit including control circuitry which cooperates with the decode unit to selectively control the decode unit to implement a precise watch or a non-precise watch on detection of a breakpoint caused by a fetched instruction having a specified program count of a fetched instruction having a specified opcode, wherein according to a precise watch, the fetched instruction causing the breakpoint and subsequent instructions are not allowed to enter the execution unit and, according to a non-precise watch, the fetched instruction causing the breakpoint and subsequent instructions are permitted to be supplied from the decode unit to the at least one execution unit. See column 9, line 48, to column 10, line 7. Note that a breakpoint/exception is caused by an illegal instruction, for instance. An illegal instruction has an opcode that is not in the set of legal opcodes. Consequently, the breakpoint is caused by a specified (illegal) opcode. In this case (precise), it and its subsequent instructions are dispatched to reservation stations of an execution unit. See Fig.2, components 50a and 50b. However, the instructions are not allowed to enter the true execution unit (execution logic) 54 for execution if a breakpoint was caused (this is determined by checking the EOK bit). It should be realized that in the claim, applicant has not defined what it means to not enter an execution unit. For purposes of this examination, the execution unit is the actual logic 54 which performs execution. It should also be noted from column 10, lines 26-28, that in one technique, the breakpoint instruction is never dispatched to the execution unit. Both of these aforementioned teachings read on applicant's claims. Furthermore, in column 10, lines 12-16, it is disclosed that the breakpoint/exception is processed after all instructions preceding the breakpoint are complete. As a result, it should be realized that the instructions subsequent to the breakpoint instruction are not executed (don't enter the execution unit) until the exception is processed. On the other hand,

note from column 22, lines 16-29, that the system also operates in a non-precise (imprecise) exception mode. More specifically, the breakpoint instruction causing the exception (caused at least in part by an instruction having a floating-point opcode) along with subsequent instructions are allowed to proceed through the pipeline so that increased performance is achieved.

e) Song has not taught that:

- the executed instructions are predicated instructions, wherein each instruction includes a guard, the value of which determines whether or not that instruction is executed. However, Grochowski has taught the concept of executing predicated instruction having guards. See Fig.6, step 610, and Fig.1 (note that p2 is a predicate/guard).
- the execution unit is associated with a guard register file holding values of the guards to allow resolution of the guards to be made. However, Grochowski has taught such a concept. See Fig.3A, component 300, and column 2, lines 65-67. Note that the predicate table (register file), is updated with actual predicate values, which are in turn used as predictions for speculatively executed instructions.
- instructions are supplied to the execution unit while guard resolution in said execution pipeline is awaited. However, Grochowski has taught such a concept. See column 3, lines 4-35. More specifically, predicated instructions are allowed to progress through the pipeline before its corresponding guard is resolved.
- issue a request to the execution pipeline for guard resolution, the guard resolution being transmitted to the control circuitry of the emulation unit which is responsive thereto to control operation of the decode unit. Note from column 3, lines 25-35,

of Grochowski, that a predicated instruction is prevented from executing until the guard is resolved. This is advantageous in that a time-expensive recovery is not required if the guard is improperly predicted. Therefore, when Song is in precise watch mode, Grochowski has taught that it is beneficial to issue a request to the execution pipeline for guard resolution for the aforementioned reasons.

A person of ordinary skill in the art would have recognized that by implementing predicated instructions with predicate prediction within Song, a) conditional branches could be eliminated, thereby reducing the amount of instructions required in the instruction set, and b) predicated instructions would be speculatively executed (i.e., executed before the corresponding guard is resolved), thereby maximizing throughput by executing predicated instructions as soon as possible. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Song to include predicated instructions and predicate prediction, as taught by Grochowski.

20. Claims 6, 13, and 17 are rejected under 35 U.S.C. 103(a) as being unpatentable over Song in view of Grochowski, as applied above, and further in view of Matt et al., EP 0943995A2 (as disclosed by applicant, applied in the previous Office Action, and herein referred to as Matt).

21. Referring to claim 6, Song in view of Grochowski has taught a computer system as described in claim 5. Song in view of Grochowski has not taught the specifics set forth in claim 6. However, Matt has taught issuing a go command and divert command to the decode unit responsive to receipt of the guard resolution from the execution pipeline, wherein a go command allows the instruction which caused the breakpoint and subsequent instructions to be normally

decoded and executed, and a divert command sets the computer system into a debug mode. See column 4, lines 6-40. More specifically, when a break event occurs, (which would include exceptions taught by Song), the system would issue either a command to resume execution (go command) or a command to execute a service routine (divert command). This allows for real-time execution control for debug functions within a processor, as taught by Matt in column 4, lines 6-9. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Song in view of Grochowski in view of Matt such that an go and divert commands may be issued in order to allow for real-time execution control for debug functions.

22. Referring to claim 13, Song in view of Grochowski has taught a method as described in claim 10.

a) Furthermore, Song in view of Grochowski has taught that in a precise watch mode, a request for guard resolution is issued such that an instruction guard is resolved prior to execution of the instruction. Note from column 3, lines 25-35, of Grochowski, that a predicated instruction is prevented from executing until the guard is resolved. This is advantageous in that a time-expensive recovery is not required if the guard is improperly predicted. Therefore, when Song is in precise watch mode, Grochowski has taught that it is beneficial to issue a request to the execution pipeline for guard resolution for the aforementioned reasons. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to issue a request to the execution pipeline for guard resolution during precise watch mode in Song.

b) Song in view of Grochowski has not taught selectively causing issue of one of a go command and a debug command responsive to the guard resolution. However, Matt has taught issuing a go command and divert command to the decode unit responsive to receipt of the guard resolution

from the execution pipeline, wherein a go command allows the instruction which caused the breakpoint and subsequent instructions to be normally decoded and executed, and a divert command sets the computer system into a debug mode. See column 4, lines 6-40. More specifically, when a break event occurs, (which would include exceptions taught by Song), the system would issue either a command to resume execution (go command) or a command to execute a service routine (divert command). This allows for real-time execution control for debug functions within a processor, as taught by Matt in column 4, lines 6-9. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Song in view of Grochowski in view of Matt such that an go and divert commands may be issued in order to allow for real-time execution control for debug functions.

23. Referring to claim 17, Song in view of Grochowski has taught a computer system as described in claim 16. Furthermore, claim 17 is rejected for the same reasons set forth in the rejection of claim 6.

24. Claims 7, 15, and 18 are rejected under 35 U.S.C. 103(a) as being unpatentable over Song in view of Grochowski, as applied above, and further in view of Adler et al., U.S. Patent No. 5,627,981 (herein referred to as Adler).

25. Referring to claim 7, Song in view of Grochowski has taught a computer system as described in claim 1. Song in view of Grochowski has not taught the specifics set forth in claim 7. However, Adler has taught a system in which the instruction which caused the breakpoint (exception) and subsequent instructions are decoded and executed normally until such time as said instruction reaches the execution pipeline where its guard is resolved such that a commit

signal is generated to the control circuitry of the emulation unit, and wherein the emulation unit is responsive to receipt of the commit signal to set the computer system into a debug mode. See the last 8 lines of the abstract and Fig.9. In essence, predicated instructions are executed speculatively (i.e., executed normally without knowing the actual value of the guard/predicate). When the instruction is at a commit point, and if the instruction's predicate is true, then if that instruction caused a breakpoint (or exception), then it will be serviced (debug mode). This scheme allows errors that should not have occurred (errors resulting from instructions that should not have executed) to be ignored while servicing the errors that should have occurred (errors resulting from instructions that have properly executed. See column 3, lines 20-42. As a result, it would have been obvious to one of ordinary skill in the art to modify the system taught by Song in view of Grochowski such that it includes the functionality of Adler.

26. Referring to claim 15, Song in view of Grochowski has taught a method as described in claim 10. Song in view of Grochowski has not taught the specifics set forth in claim 15. However, Adler has taught a system in which the instruction which caused the breakpoint (exception) and subsequent instructions are decoded and executed normally until such time as the guard is resolved wherein, if the guard is resolved such that a position commit signal is generated, the processor is set into a debug mode. See the last 8 lines of the abstract and Fig.9. In essence, predicated instructions are executed speculatively (i.e., executed normally without knowing the actual value of the guard/predicate). When the instruction is at a commit point, and if the instruction's predicate is true, then if that instruction caused a breakpoint (or exception), then it will be serviced (debug mode). This scheme allows errors that should not have occurred (errors resulting from instructions that should not have executed) to be ignored while servicing

the errors that should have occurred (errors resulting from instructions that have properly executed. See column 3, lines 20-42. As a result, it would have been obvious to one of ordinary skill in the art to modify the system taught by Song in view of Grochowski such that it includes the functionality of Adler.

27. Referring to claim 18, Song in view of Grochowski has taught a computer system as described in claim 16. Furthermore, claim 18 is rejected for the same reasons set forth in the rejection of claim 7.

28. Claim 11 is rejected under 35 U.S.C. 103(a) as being unpatentable over Song in view of Grochowski, as applied above, and further in view of Kurakazu, U.S. Patent No. 5,644,703 (as disclosed by applicant, applied in the previous Office Action, and herein referred to as Kurakazu).

29. Referring to claim 11, Song in view of Grochowski has taught a method as described in claim 10. Song in view of Grochowski has not explicitly taught that breakpoints are detected at instructions having certain program counts. However, Kurakazu has taught that an address break is well known, accepted, and expected in the art. See column 1, lines 12-20. More specifically, a user is allowed to set a breakpoint at a specified address in order to determine the status of the system at that particular point regardless of the instruction that is at the address. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Song in view of Grochowski such that breakpoints are detected at instructions having certain program counts (addresses).

Response to Arguments

30. Applicant's arguments filed on October 19, 2005, have been fully considered but they are not persuasive.

31. Applicant argues the novelty/rejection of claim 1 on pages 7-8 of the remarks, in substance that:

> "While it is true that some predicated processors may not require the use of branch instructions and thus the number of instructions in the instruction set is reduced, this benefit may be offset by the fact that every other instruction includes additional bits for the predicate field that are not required in non-predicated processors. Thus, while there may be fewer types of instructions in a predicated processor (i.e., because branch instructions are not used) the total amount of code in a program may actually be increased due to the fact that additional bits for the predicate field must be included in every other instruction."
>
> "Applicants further disagree that a predicated processor maximizes throughput by executing predicated instructions as soon as possible. In predicated processors, many instructions may make it most of the way through the execution pipeline whether they are to be executed or not. Thus, for instructions whose predicate is eventually resolved as false, many execution cycles may have been wasted processing the instruction in the pipeline, even though the instruction will never be executed. Applicants respectfully disagree that this maximizes processor throughput."
>
> "Further, the processor of Song processes multiple instructions simultaneously and executes instructions out of order, relative to their programmed sequence. That is, Song discusses that in some situations it may be necessary to delay execution of certain instructions because the operands of these instructions may depend on the result of previous instructions that have not yet completed execution (Col. 5, lines 40-54). Thus, it may not be preferential or even possible in the system of Song to execution instructions "as soon as possible," as the instruction should be delayed to wait for the result of another instruction."
>
> "Moreover, because the processor of Song is an out-of-order processor that executes multiple instructions at the same time using multiple execution units and relies on a sequencer unit to determine the order and timing in which instructions should be sent to the various execution units, it would be difficult, if not impossible, to incorporate predicates into the system of Song without upsetting the timing and synchronization that is required in executing multiple instructions at the same time, using different execution units."

32. These arguments are not found persuasive for the following reasons:

a) Regarding the first argument, the examiner has given one possible benefit of implementing predication. Some of ordinary skill in the art would be willing to gain this benefit even at the cost of increasing program size (as applicant argues), as the benefit may be more important for that particular system. For instance, perhaps, instruction word size may be decreased with the

elimination of conditional branch instructions. Or, additional instructions could be implemented which would replace the branches. Either way, this is a motivation for combination. The fact that a drawback may occur does not mean that the system cannot be combined in order to achieve the benefit.

b) Regarding the second argument, predicate prediction, which is known, is done to improve throughput. Otherwise, there would be no reason to implement it. Granted, predicate prediction does not achieve maximum theoretical throughput (nothing does), but if a predicate system has a throughput of X, then the addition of predicate prediction to the same predicate system, will maximize the throughput as much as a prediction system can. The increase in throughput is directly related to the accuracy of the predictions.

c) Regarding the third argument, an instruction may be delayed until its operands are ready at time X, but time X may still be before time Y, which is the normal time at which the instruction should be executed. An instruction may be executed as soon as its operands are ready, or as soon as possible.

d) Regarding the fourth argument, applicant is merely alleging that such a combination is impossible or difficult. There is inherently some complexity involved with combining any two systems. However, this does not necessarily preclude combination. U.S. Patents 5,471,593 and 6,009,512 are two examples of out-of-order machines which employ predication. The combination has been done before, and the examiner has given reasons why it is beneficial.

33. Applicant argues the novelty/rejection of claim 1 on pages 10 of the remarks, in substance that:

> "...to the extent that Song discusses preventing execution of breakpoint instructions at all, this function is not performed by an emulation unit including control circuitry which cooperates with the decode unit, as required by claim 1. There is no teaching or suggestion that this function can be performed by an emulation unit (i.e., a unit used for diagnostic or debugging purposes)"

34. These arguments are not found persuasive for the following reasons:

a) The examiner asserts that the prior art has taught emulation circuitry since it essentially operates in the same manner as applicants' circuitry. To debug is to find and remove errors or bugs in a program. The circuitry of Song does just this as it detects exceptions (errors) and corrects them. Hence Song does a form of run-time debugging even though Song doesn't explicitly use the word debugging or emulation in the specification. However, the functionality is the same.

35. Applicant argues the novelty/rejection of claim 1 on pages 10 of the remarks, in substance that:

> "...to the extent that Song discusses preventing execution of breakpoint instructions at all, this function is not performed by an emulation unit including control circuitry which cooperates with the decode unit, as required by claim 1. There is no teaching or suggestion that this function can be performed by an emulation unit (i.e., a unit used for diagnostic or debugging purposes)"

36. These arguments are not found persuasive for the following reasons:

a) The examiner asserts that the prior art has taught emulation circuitry since it essentially operates in the same manner as applicants' circuitry. To debug is to find and remove errors or bugs in a program. The circuitry of Song does just this as it detects exceptions (errors) and corrects them. Hence Song does a form of run-time debugging even though Song doesn't explicitly use the word debugging or emulation in the specification. However, the functionality is the same.

37. Applicant argues the novelty/rejection of claim 1 on pages 11-12 of the remarks, in substance that:

> "The Office Action asserts that Song teaches this limitation at column 9, line 48 - column 10, line 7 and column 22, lines 16-29. Applicants respectfully disagree. The cited paragraph of Song teaches that for any instruction that causes an exception, the instruction causing the exception is not executed. Song does not teach or suggest that subsequent instructions are not allowed to enter the execution unit...The processor of Song is capable of execution multiple instructions simultaneously using different execution units. Nowhere does Song teach or suggest that an exception causes the pipeline to be halted (with respect to instructions not yet in the pipeline) until the exception is processed."

38. These arguments are not found persuasive for the following reasons:

a) The examiner wants to first point out that the claim calls for subsequent instructions not being allowed to enter the execution unit. The claim does not say that <u>all</u> subsequent instructions are not to enter the execution unit. Consequently, even just two of X subsequent instructions not entering the execution unit would read on the claim. The examiner asserts that if the exception causing instruction is not executed, then instruction dependent on the exception-causing instruction cannot be executed until the exception is fixed. Clearly, if subsequent instructions need to utilize a result of the exception-causing instruction, then they cannot execute until that result is ready. This is exactly the purpose of the reservation station(s) of Song. A reservation station, as is known in the art, buffers instructions until their operands are ready. Once they are, the instruction can be issued. In this case, if the exception-causing instruction does not finish, then some instructions will not have access to their operands, thereby preventing execution of those instructions (they will be held in the reservation station until the exception is fixed and the operands are ready). These dependent instructions are subsequent instructions. Even if they don't make up the entire set of subsequent instructions, the claim does not require that. Only two or more subsequent instructions need to be prevented from executing, and Song has taught this.

39. The rejections of claims 10 and 16 have been argued in a similar fashion as above. Consequently, the examiner's responses above also apply to these arguments.

Conclusion

40. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to David J. Huisman whose telephone number is (571) 272-4168. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

DJH
David J. Huisman
January 6, 2006

Eddie Chan

EDDIE CHAN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100